Accelerated Data Management Systems Through Real-Time Specialization

Anastasia Ailamaki

for Periklis Chrysogelos, Aunn Raza, Panos Sioulas, and the DIAS team
The game changers

**Hardware**

Heterogeneous and underutilized

**Workloads**

Complex and unpredictable
Agile data management engines

Where’s the sweet spot?

Portability

Performance

Heterogeneity Oblivious

Hardware Oblivious

Hardware Conscious

Heterogeneity Conscious

+ALP
Device specialization carries portability debt

Performance

Hardware Features

HW-promised performance

Specialization benefit

Portability debt

XXX-conscious engine

YYY-conscious engine

HW-oblivious engine
Analytics on heterogeneous hardware

Device specialization carries portability debt

Total Execution Time (s)

CPU-only  Hybrid  GPU-only

Join-heavy  50%-50% Mix  Scan-heavy

2x Intel(R) Xeon(R) Gold 5118 CPU
2x Mellanox MT27800 100G IB NIC
2x NVIDIA Tesla V100S GPU
13 Queries, CPU-resident data
96-144GB working set/query
Join-heavy: SPJ{1-4}Aggr
Scan-heavy: Scan-Aggr
4 servers
Lifetime of a Query

SELECT SUM(T3.c * T4.d) 
FROM T1, T2, T3, T4, ...
WHERE T1.f < 50 AND T1.a = T2.t1_id AND ...

Query Execution Plan

Query Parsing & Optimization

Control-heavy

Sequential

Random

Result

SUM 5
Hardware-conscious Analytics?

**Traditional**
- Random-access
- Control-heavy
- Sequential scan

**CPU-optimized**
- radix-(join/group by)
- vector-at-a-time
- parallelism/interocket atomics

**Relies on**
- High cache-size-to-thread ratio
- Efficient inter-socket operations

**Won’t work on GPUs**
A fast equi-join algorithm

Radix-join
- Partition both inputs
- Size partition fanout based on memory hierarchy (TLB+caches)
- Assuming sufficient cache-to-thread ratio

GPU memory hierarchy
- Low cache-to-thread ratio
- Software and hardware-managed caches
- But collaborative thread execution

Think differently for GPUs!
Collaboratively partition per GPU thread block
- Amortize radix cluster maintenance
- Rely on big register files and thread overlapping
- Avoid random accesses to GPU memory

Stage partition output in scratchpad
- Irregular access patterns through scratchpad
- Coalesce writes through shared memory
- Multiple threads “complete” a cache line

3.6x speedup
## Accelerator-conscious Analytics

### Traditional
- Random-access
- Control-heavy
- Sequential scan

### CPU-optimized
- radix-(join/group by)
- vector-at-a-time
- parallelism/inter-socket atomics

### CPU-GPU
- Tune operators to memory hierarchy specifics
- Code fusion & specialization for fast composability
- Encapsulate heterogeneity and balance load
SELECT SUM(a) 
FROM T 
WHERE b > 42
JIT Code Generation for OLAP in GPUs

[VLDB2019]

```
def unpack_filter_reduce(data_block, N, state):
    local_acc ← 0
    for i = threadIdInWorker to N - 1 with step #threadsInWorker
        t ← data_block[i]
        if t.a > 42
            local_acc ← local_acc + t.b
        nh_acc ← neighborhood_reduce(local_acc)
        if thread neighborhood leader
            atomic_add(state.acc, nh_acc)
```
Device providers

Inject target-specific info
From SQL to Pipeline Orchestration

```
SELECT SUM(a)
FROM T
WHERE b > 42
```

Logical plan

- aggregate
- filter
- scan

HetExchange

Multiple pipeline instances

pipeline id
- x = GPU pipeline
- = CPU pipeline

instances

1.aggregate
2.router
3.gpu2cpu
4.aggregate
5.filter
6.unpack
7.cpu2gpu
8.mem-move
9.router
10.segmenter

routing point

device crossing

JIT

Run

pipeline id
- x = GPU pipeline
- = CPU pipeline

instances

11.routing point

pipeline id
- x = GPU pipeline
- = CPU pipeline

instances

12.routing point

pipeline id
- x = GPU pipeline
- = CPU pipeline

instances

13.routing point

pipeline id
- x = GPU pipeline
- = CPU pipeline

instances

14.routing point

pipeline id
- x = GPU pipeline
- = CPU pipeline

instances

15.routing point

pipeline id
- x = GPU pipeline
- = CPU pipeline

instances

16.routing point

pipeline id
- x = GPU pipeline
- = CPU pipeline

instances
JIT data flow inspection

Decouple data- from control-flow
Encapsulate trait conversions into operators
Inspect flows to load-balance

<table>
<thead>
<tr>
<th>Flow</th>
<th>Scope</th>
<th>Trait</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>Delegation</td>
<td>Heterogeneous Parallelism</td>
</tr>
<tr>
<td></td>
<td>Routing</td>
<td>Homogeneous Parallelism</td>
</tr>
<tr>
<td>Data</td>
<td>Transfer</td>
<td>Data Locality</td>
</tr>
<tr>
<td></td>
<td>Granularity</td>
<td>Execution Granularity</td>
</tr>
</tbody>
</table>

Distribute load to devices adaptively

[VLDB2019]
Abstractions for fast CPU-GPU analytics

**Intra-operator**
- Operator tuning is μ-architecture specific
- Tune operators to memory hierarchy specifics

**Intra-device**
- Portability clashes with specialization
- Inject target-specific info using codegen

**Inter-device**
- Limited device inter-operability
- Encapsulate heterogeneity and balance load

Selective obliviousness

[CIDR2019]
The game changers

Hardware

- Heterogeneous and underutilized

Workloads

- Complex and unpredictable
Specialized OLTP & OLAP Systems

Data freshness bounded by ETL latency
Hybrid Transactional and Analytical Processing

OLTP: task-parallel
- High rate of short-lived transactions
- Mostly point accesses (high data access locality)

OLAP: data-parallel
- Few, but long-running queries
- Scans large parts of database

Align tasks & hardware to improve utilization
HTAP: Chasing ‘locality of freshness’

Static OLAP-OLTP assignment
- Unnecessary tradeoff between interference and performance
- Pre-determined resource assignment based on workload type
- Wasteful data consolidation and synchronization

Real-time, Adaptive scheduling of HTAP workloads
- Specialize to requirements and data/freshness-rates
- Workload-based resource assignment
- Pay-as-you-go snapshot updates

Task placement based on resource usage
Workload Isolation & Fresh Data Throughput

Interference ↔ performance

Pre-determined resource assignment

Isolated

Hybrid-Access

Elastic-Compute

Colocated

Fresh Data Access Bandwidth

Independent execution (isolation)

no extreme is good
Workload Isolation & Fresh Data

Isolated

Hybrid-Access

Elastic-Compute

Colocated

Real-time: Adaptive scheduling of HTAP workloads

- Specialize to requirements and amount of unconsolidated data
- Workload-based resource assignment
- Pay-as-you-go snapshot updates

Task placement & consolidation based on
Caldera: HTAP on CPU-GPU Servers

Store data in shared memory

Run OLTP workloads on task-parallel processors

Run OLAP workloads on data-parallel processors

– On-demand copy-on-write snapshots in shared memory
GPU Accesses Fresh Data from CPU Memory

OLTP generates fresh data on CPU Memory

Data access protected by concurrency control

OLAP needs to access fresh data

Snapshot isolation for OLAP w/o CC overheads

[CIDR2017, CIDR2020, SIGMOD2020]
GPU Accesses Fresh Data from CPU Memory

OLTP generates fresh data on CPU Memory

Data access protected by concurrency control

OLAP needs to access fresh data

snapshot isolation for OLAP w/o CC overheads
Increasing workload complexity

Diverse modern data problems
  – IOT, OCR, ML, NLP, Medical, Mathematics etc...

DBMS catch-up for popular functionality
  – Human effort and big delays
  – Oblivious to out-of-DBMS workflows

Vast resource of libraries
  – Authored by domain experts, used by everybody
  – Loose library-to-data-sources integration and optimization

Need for systems that can “learn” new functionality
Network looks like a single machine

Similar intra-/inter-server interconnect bandwidth

Local memories and NUMA effects across devices

CPU-GPU: Capacity-Throughput
A solution is only as efficient as its least adaptive component.